Cocoa Scripting

Session 303

Cocoa Scripting

Mark Piccirelli
Cocoa Frameworks Group

What You'll Learn

- The basics of Cocoa Scripting, if you have not used it yet
- What scripting commands and classes are already provided by Cocoa
- How to declare new scripting classes
- How to implement scripting classes

What You'll Learn (Cont.)

- The types you can use in scripting suites
- New features
- Future directions

Cocoa Scripting Basics

- AppleScript presents scriptability as *commands* and *classes* that are grouped into *suites*
- Applications handle commands
- Applications expose instances of scriptable classes

Cocoa Scripting Basics

What Cocoa does for you

- Gives you a way to declare commands and classes
- Handles incoming Apple events, converts them to command objects, and executes the command objects
- Declares and implements most of the Standard Suite and Text Suite commands and classes

Cocoa Scripting Basics

What your application must do

- Include an NSAppleScriptEnabled entry in its Info.plist
- Include files in its bundle to declare scripting suites
- Provide methods to handle custom commands
- Provide accessor methods in scriptable classes

Files in the App's Bundle

.scriptSuite and .scriptTerminology files

- Parallel property lists
- .scriptSuite declares scripting model and mappings to Objective-C classes and methods
- .scriptTerminology contains human-readable names and descriptions
- Cocoa automatically parses them both and creates 'AETE' data at run time

Files in the App's Bundle

.scriptSuite and .scriptTerminology files

- Each contains a Commands dictionary
- Each contains a Classes dictionary
- .scriptSuite also contains AppleEventCode and (programmatic) Name entries
- .scriptTerminology also contains human-readable Description and Name entries
- Other stuff to support custom value types

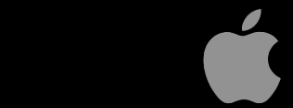
Commands

Declare them in the .scriptSuite file

- AppleEventClassCode and AppleEventCode
- Arguments
- AppleEventCode and Type for each argument
- ResultAppleEventCode
- Type (of the return value)
- CommandClass

CommandClass?

- The name of an Objective-C subclass of NSScriptCommand (or NSScriptCommand itself)
- Scripted operations are sent to scriptable applications via Apple events
- Cocoa converts each scripting Apple event to an instance of the CommandClass, and then sends the command object an -executeCommand message
- Subclassing NSScriptCommand often is not necessary



Commands

Declare them in the .scriptTerminology file too

- Human-readable Description
- Human-readable Name
- Arguments—the exact same set as in the .scriptSuite file
- Human-readable Description and Name for each argument

Example

The Close command

- Part of the AppleScript Standard Suite
- Declared in the Foundation framework's NSCoreSuite.scriptSuite and NSCoreSuite.scriptTerminology files
- Encapsulation and default implementation provided by Foundation's NSCloseCommand class
- AppKit's NSDocument knows how to handle it

Built-in Commands

Cocoa takes care of the basics

- The Standard Suite declares commands: open, print, quit, close, save, count, delete, duplicate, exists, make, move, etc.
- Foundation and AppKit provide implementations for all of these—NSCountCommand, NSDeleteCommand, etc.
- Foundation provides implementations of the all-important get and set commands too—NSGetCommand and NSSetCommand

Built-in Commands

Cocoa takes care of the basics

- Your scriptable classes have to work with Foundation's standard command classes
- Key Value Coding
- You shouldn't have to declare many new commands

Classes

Declare them in the .scriptSuite file

- Superclass
- AppleEventCode
- Attributes (correspond to AppleScript properties)
- ToManyRelationships (elements)
- AppleEventCode, ReadOnly, and Type for each attribute and to-many relationship
- SupportedCommands
- (ToOneRelationships)

Classes

Declare them in the .scriptSuite file

- Class name should virtually always be an Objective-C class name, e.g., “NSWindow”
- Superclass name should be name of class in same suite, or suite-specified name of class in other suite e.g., “NSCoreSuite.NSWindow”

Classes

Declare them in the .scriptTerminology file too

- Human-readable Description
- Human-readable Name and PluralName
- Attributes—the exact same set as in the .scriptSuite file
- Human-readable Description and Name for each attribute

Example

A page setup class

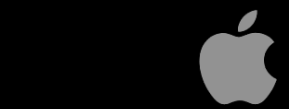
- TextEdit document page setup should be scriptable
- Let’s look at the declarations to add it

Demo

Built-in Classes

Cocoa takes care of the basics

- Standard Suite classes: application, document, window
- Text Suite classes are implemented by NSTextStorage
- Subclassing just to add scripting properties and elements is unnecessary with Objective-C; use categories



Command Execution

What's in an NSScriptCommand?

- Virtually everything from the Apple event, but in a different form
- An object specifier for the receiver(s) of the command
- Arguments, some of which might also be object specifiers, some of which might be value objects
- Object specifiers are instances of subclasses of NSScriptObjectSpecifier

Object Specifiers

NSScriptObjectSpecifier and its subclasses

- AppleScript references:
  word 4 of text of front document
- Become NSScriptObjectSpecifer chains:

  NSIndexSpecifier (word 4) →

  NSPropertySpecifier (text) →

  NSIndexSpecifier (document 1)

Object Specifiers

NSScriptObjectSpecifier and its subclasses

- Cocoa has an NSScriptObjectSpecifier subclass for every AppleScript reference form:
- Index reference ➡ NSIndexSpecifier
- Property Reference ➡ NSPropertySpecifier
- Range reference ➡ NSRangeSpecifier
- Name reference ➡ NSNameSpecifier
- Etc.

Object Specifiers

NSScriptObjectSpecifier and its subclasses

- Object specifiers are evaluated during command execution
- Actual objects are found
- Containers are asked for attribute values and relationship objects

Command Execution

What are the steps?

- Evaluation of the receivers object specifier
- Evaluation of any argument object specifiers
- Send the command to each receiver if they handle it (according to SupportedCommands in the receiver class' .scriptSuite declaration)
- More likely, invoke the script command's -performDefaultImplementation method

Command Execution

How do Foundation's commands work?

- Most of Foundation's NSScriptCommand subclasses override --performDefaultImplementation
- Your classes don't have to explicitly support these commands (get, set, make, move, etc.)
- Make your classes conform to Key Value Coding, and we'll do the rest

Key Value Coding (KVC)

- KVC lets us invoke methods in your class, without knowing their names
- KVC lets you write your scripting support methods in a natural way
- NSScriptObjectSpecifier uses KVC to find specified objects during evaluation
- NSGetCommand, NSSetCommand, NSMoveCommand, etc. all use KVC
- A great use of Objective-C's dynamism

KVC for Scriptable Attributes

Implement get-accessors in classes with attributes

- Your objects will be sent -valueForKey:*attributeName* messages
- *attributeName* will be the string used to declare the attribute in the .scriptSuite file
- You could override -valueForKey:
- Better yet, implement *-attributeName* methods; -[NSObject valueForKey:] will invoke them
- Return an object of the type declared for the attribute in the .scriptSuite file

KVC for Scriptable Attributes

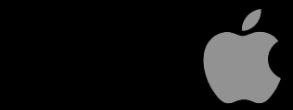
Set-accessors in classes with writable attributes

- Your objects will be sent -takeValue:value forKey:attributeName messages for setting of non-read-only attributes
- Implement -setAttributeName: methods
- The object's type isn't guaranteed, so check

Example

A page setup class

- Let's look at the code to implement the scriptability we've already declared



Demo

KVC for To-Many Relationships

Get-accessors in classes that contain elements

- Your objects will be sent -valueForKey:*relationshipName* messages
- *relationshipName* will be the string used to declare the relationship in the .scriptSuite file
- Implement *-relationshipName*: methods
- Return an array of objects

KVC for To-Many Relationships

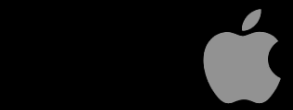
Get-accessors in classes that contain elements

- Your objects may be sent -valueAtIndex:*index* inPropertyWithKey:*relationshipName* messages
- Implement -valueIn*RelationshipName*AtIndex: methods
- Return a single object
- Potentially much more efficient than creating an array to return from -relationship*Name:* methods

KVC for To-Many Relationships

Get-accessors in classes that contain elements

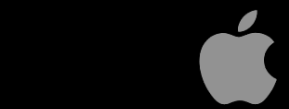
- There are now -valueWithName:inPropertyWithKey: and -valueWithUniqueID:inPropertyWithKey: messages too
- Implement -valueIn*RelationshipName*WithName: and -valueIn*RelationshipName*WithUniqueID: methods
- Return a single object
- Used for elements that have name and ID properties



KVC for To-Many Relationships

Set-accessors in classes that contain elements

- Three more possible messages:
  - -replaceValueAtIndex:inPropertyWithKey:withValue:
  - -insertValue:atIndex:inPropertyWithKey:
  - removeValueAtIndex:fromPropertyWithKey:
- You implement:
  - -replaceIn*RelationshipName*:atIndex:
  - -insertIn*RelationshipName*:atIndex:
  - -removeIn*RelationshipName*:atIndex:



Initialization for Element Classes

- Some classes can be instantiated with the Make command
- Make sure script-creatable classes have good -init methods
- Make sure script-creatable classes have good attribute set-accessors, so that Make's "with properties" argument works
- Make sure possible containers for such classes have -replaceIn*RelationshipName*:atIndex: and -insertIn*RelationshipName*:atIndex: methods

Object Specifier Support

- Your scriptable object may be sent an -objectSpecifier message
- Right after initialization, during execution of a Make command
- As the result of a Get command
- Return an instance of one the subclasses of NSScriptObjectSpecifier; e.g. NSIndexSpecifier, NSNameSpecifier, NSUniqueIDSpecifier
- Might have to ask the container for its object specifier

Types in .scriptSuite Files

- Class attributes, class relationships, command arguments, command results all must have declared types
- Scriptable class names
- Selected Foundation class names
- Some more complicated things

Types in .scriptSuite Files

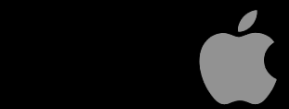
Scriptable class names

- Names of classes declared in the same .scriptSuite file are fine
- So are classes declared in other .scriptSuite files; suite-specify the name like this: *suiteName.className*
- You can use the classes declared in Foundation's NSCoreSuite

Types in .scriptSuite Files

Selected Foundation class names

- NSString — AppleScript “unicode text”
- NSDate — “date”
- NSArray (big bug fixed!) — “list”
- NSDictionary (support for user records is improved) — “record”
- NSScriptObjectSpecifier — “reference”
- NSPositionalSpecifier — “location reference”



Types in .scriptSuite Files

Different kinds of NSNumber

- E.g., NSNumber<Bool> — “boolean”
- NSNumber<Int> — “integer”
- NSNumber<Double> — “real”
- Etc.

Types in .scriptSuite Files

One more kind of NSNumber

- NSNumber<*enumerationName*>
- Supports AppleScript enumeration types, e.g., for the Close command's saving parameter—"ask, yes, no"
- .scriptSuite and .scriptTerminology files have Enumerations dictionaries too

NSNameSpecifier

New!

- Supports AppleScript's name reference form
- E.g., **window named "Cocoa Scripting"**
- Must be an attribute with an AppleEventCode of 'pnam'
- Your container class should have -valueIn*RelationshipName*WithName:
- Often the fastest way to find one of a large number of named elements

NSUniqueIDSpecifier

New!

- Supports AppleScript's ID reference form
- E.g., **window 783**
- Must be an attribute with an AppleEventCode of 'ID '
- Your container class should have -valueIn*RelationshipName*WithUniqueID:

The Properties Property

New!

- Good scriptable apps make all object properties available in a “properties” record
- You don’t have to do anything
- Public -scriptingProperties and -setScriptingProperties: methods, just in case

Implicitly Specified Subcontainers

New!

- Used to have to write:
  fourth word of text of front document
- Better to let scripters write:
  fourth word of front document
- New DefaultSubcontainerAttribute entry allowed in .scriptSuite class descriptions

NSAppleEventDescriptor

New methods

- A few places where NSAppleEventDescriptor shows in Cocoa's API
- New creation methods like +descriptorWithBool:, +descriptorWithString:, etc.
- New accessor methods like -boolValue, -stringValue, etc.

NSAppleScript

New!

- At last, execute scripts without using Carbon API
- Can initialize from file, including compiled scripts
- Can initialize from source code
- Compile, execute, etc.
- Can even return source code as pretty-printed rich text

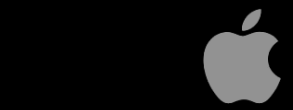
Release Notes

- Very Detailed!
- Cocoa Scripting release notes are on your Jaguar CD

/Developer/Documentation/ReleaseNotes/Foundation.html

Future Directions

- Start using AppleScript's new XML-based .sdef suite definition format
- Even before then, more attractive scripting dictionaries (categories, orders)
- Better type checking
- Much better error reporting



Future Directions

- Cocoa API for sending Apple events
- Better integration with undo
- Stop asking objects for their object specifiers; ask objects' *containers* instead
- Recordability, recordability, recordability . . .

Roadmap

106 AppleScript Update General scripting news	Room C **Wed., 9:00am**
902 AppleScript Studio Intro A big user of Cocoa Scripting	Civic Center **Wed., 3:30 pm**
FF007 AppleScript: General scripting issues	Room J1 **Thurs., 3:30 pm**
FF016 Cocoa: Cocoa-specific issues	Room A1 **Fri., 5:00 pm**

Who to Contact

Heather Hickman
Cocoa Technology Manager
hhickman@apple.com

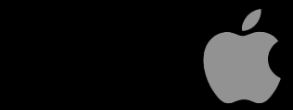
http://developer.apple.com/wwdc2002/urls.html

Documentation

Designing a scriptable application

- Topic: Application Architecture
  - Overview, concepts
  - Links to related class reference

Documentation >Mac OS X > Cocoa > Program Design
http://developer.apple.com/techpubs/macosx/Cocoa/CocoaTopics.html

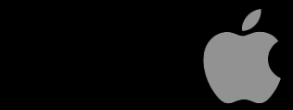


Documentation

Implementing a scriptable application

- Topic: Scriptable Applications
  - Overview, concepts, programming tasks
  - Links to scripting class reference
  - Links to other AppleScript documentation

Documentation >Mac OS X > Cocoa > Program Design
http://developer.apple.com/techpubs/macosx/Cocoa/CocoaTopics.html



Documentation

An oldie but goody

- "Designing a Scripting Implementation" by Cal Simone—Develop Issue 21 (March 1995!)
- Great guidelines for designing an app's scriptability
- Some advice about four-character codes

http://developer.apple.com/dev/techsupport/develop/bysubject/iac.html

For More Information

- Cocoa Developer Documentation
  http://developer.apple.com/techpubs/macosx/macosx.html
- iServices Technical Training
  http://www.apple.com/iservices/technicaltraining
- O'Reilly Network: AppleScripting Mac OS X
  http://www.oreillynet.com/pub/ct/47
- Other places
  - **www.stepwise.com**
  - **www.omnigroup.com**
  - **www.cocoadevcentral.com**

SuiteModeler

by Don Briggs

- A neat shareware app for editing .scriptSuite and .scriptTerminology files, among other things
- Good at flagging mistakes and conflicts
- Check out the EZCocoaAppleScript tutorial too

http://homepage.mac.com/donbriggs

Q&A


Heather Hickman
Cocoa Technology Manager
hhickman@apple.com

http://developer.apple.com/wwdc2002/urls.html


WWDC2002


WWDC2002


WWDC2002